Learning All Optimal Policies with Multiple Criteria

Leon Barrett BARRETT@ICSI.BERKELEY.EDU
1947 Center St. Ste. 600, Berkeley, CA 94704

Srini Narayanan SNARAYAN@ICSI.BERKELEY.EDU
1947 Center St. Ste. 600, Berkeley, CA 94704

Abstract

We describe an algorithm for learning in the presence of multiple criteria. Our technique generalizes previous approaches in that it can learn optimal policies for all linear preference assignments over the multiple reward criteria at once. The algorithm can be viewed as an extension to standard reinforcement learning for MDPs where instead of repeatedly backing up maximal expected rewards, we back up the set of expected rewards that are maximal for some set of linear preferences (given by a weight vector, $\overrightarrow{w}$). We present the algorithm along with a proof of correctness showing that our solution gives the optimal policy for any linear preference function. The solution reduces to the standard value iteration algorithm for a specific weight vector, $\overrightarrow{w}$.

1. Introduction

In Reinforcement Learning (RL), an agent interacts with the environment to learn optimal behavior. (Sutton & Barto, 1998) Most RL techniques are based on a scalar reward, i.e., they aim to optimize an objective that is expressed as a function of a scalar reinforcement. A natural extension to traditional RL techniques is thus the case where there are multiple rewards. In many realistic domains, actions depend on satisfying multiple objectives simultaneously (such as achieving performance while keeping costs low, a robot moving efficiently toward a goal while being close to a recharging station, or a government funding both military and social programs). Learning optimal policies in many real-world domains thus depends on the ability to learn in the presence of multiple rewards. However, the resulting policies depend heavily on the preferences over these rewards, and they may change swiftly as preferences vary. We present both an algorithm for the general case of learning all optimal policies under all assignments of linear priorities for the reward components, and a proof showing the correctness of our algorithm.

We start with a motivating example of a simple task with multiple rewards in Section 2. The paper then proceeds to the main algorithm in Section 3. We address related work in Section 4, and then Section 5 discusses the complexity of our algorithm including realistic and tractable specializations of our algorithm. Section 6 describes the application of this algorithm to an example domain, and Section 7 discusses extensions to this technique, such as implementations using other RL methods (such as temporal difference methods) and applications of our algorithm to infer another agent's preferences based on observing their behavior. Section 8 outlines the proof of the algorithm's correctness.

2. Explanation and Motivating Example

We assume that instead of getting a single reward signal, the agent gets a reward divided up

Appearing in *Proceedings of the 25th International Conference on Machine Learning*, Helsinki, Finland, 2008. Copyright 2008 by the author(s)/owner(s).

into several components, a reward vector. That is, we decompose the reward signal $r(s,a)$ (where s is a state and a is an action) into a vector $\overrightarrow{r}(s,a) = [r_1(s,a), r_2(s,a), \ldots, r_n(s,a)]$. An agent could potentially optimize many different functions of these rewards, but the simplest function is a weighted sum: for every fixed weight vector $\overrightarrow{w}$ we obtain a total reward scalar $r_{\overrightarrow{w}}(s,a) = \overrightarrow{w} \cdot \overrightarrow{r}(s,a)$. There is thus an optimal policy $\pi^*_{\overrightarrow{w}}$ for *each* weight vector $\overrightarrow{w}$.

Consider, for example, a lab guinea pig running a familiar maze, shown in Figure 1. The guinea pig runs through the maze to one of four stashes of food. Once it has reached a stash and eaten the food, the experimenter takes it out of the maze and returns it to its cage, so it can only hope to eat one of the stashes per run of the maze. Assume that there are only 2 types of food provided (hay and carrot), so reward vectors take the form [hay,carrot]. Location 1 contains hay ($\overrightarrow{r} = [1,0]$), location 2 contains carrot ($\overrightarrow{r} = [0,1]$), and locations 3 and 4 contain a little of both ($\overrightarrow{r} = [0.6, 0.6]$ and $[0.7, 0.4]$, respectively). Because the maze is familiar, the animal knows where the food is placed and what sort of food is in each location.

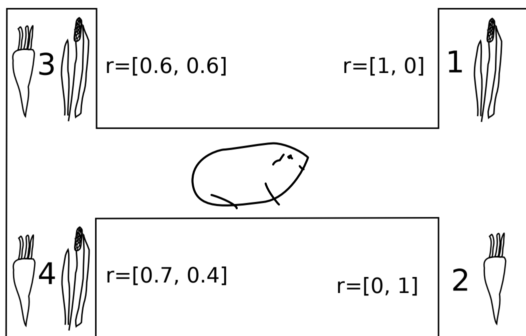

Figure 1. An example maze with rewards, split into 2 components, at 3 different locations

The experimenter has several different guinea pigs and has discovered that each has different tastes. For instance, Chester likes only hay ($\overrightarrow{w} = [1,0]$), and Milo likes only carrot ($\overrightarrow{w} = [0,1]$), but greedy Louis likes both equally ($\overrightarrow{w} = [0.5, 0.5]$). (Without loss of generality, assume that all animals' weight vectors satisfy $\sum_i w_i = 1$: they describe relative preferences, not absolute utilities.) So, if Chester goes to location 4 ($\overrightarrow{r} = [0.7, 0.4]$), then he gets reward $r = \overrightarrow{w} \cdot \overrightarrow{r} = 0.7$. Milo would get 0.4, and Louis would get a reward of 0.55.

Looking at the maze, we see that although there are 4 possible strategies (with rewards shown in Figure 2, only 3 of them are optimal for any values of $\overrightarrow{w}$. One strategy occurs when the weight vector has $w_0 > 0.6$ (and hence $w_1 = 1 - w_0 < 0.4$): then the guinea pig should head straight for location 1, because the reward elsewhere will be no more than 0.6. By the exact same logic, when the weight vector has $w_1 > 0.6$ (and $w_0 < 0.4$), then the animal should go to location 2. In all other cases ($0.4 \leq w_0 \leq 0.6$), it will optimize its reward by going to location 3. Under no circumstances would an optimal agent go to location 4! No matter what its weight vector, some other location dominates location 4. We would like to determine exactly this: which policies are viable and which are not (even without knowing $\overrightarrow{w}$).

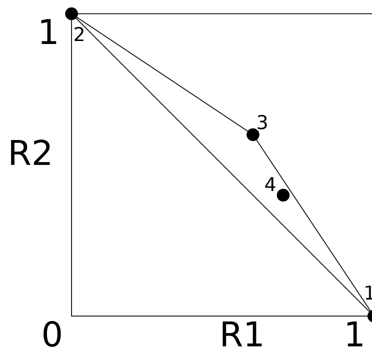

Figure 2. The potential reward vectors in the guinea pig example

Our method learns the set of optimal policies for *all* $\overrightarrow{w}$ at the same time. Once the agent has learned all these policies, it can change reward weights at runtime to get a new optimal behavior, without having to do any relearning. For a fixed priority scheme (fixed weight vector $\overrightarrow{w}$) over the multiple reward components, our algorithm results in the standard recurrence for Q-values that is analogous to the equation for the average weighted reward case as in (Natarajan & Tadepalli, 2005):

$$Q^*_{\overrightarrow{w}}(s,a) = \mathbb{E}\left[\overrightarrow{w} \cdot \overrightarrow{r}(s,a) + \gamma \max_{a'} Q^*_{\overrightarrow{w}}(s', a') | s, a\right]$$

In the general case, where we do not know the relative priorities over the reward components, our

algorithm exploits the fact that the extrema of the set of Q-values vectors (Q vectors that are maximal for some weight setting) is the same as the *convex hull* of the Q-value vectors. (The convex hull is defined as the smallest convex set that contains all of a set of points. In this case, we mean the points that lie on the boundary of this convex set, which are of course the extreme points–the ones that are maximal in some direction. This is somewhat similar to the Pareto curve, since both are maxima over trade-offs in linear domains.) Now we can rewrite the general RL recurrence in terms of operations on the convex hull of Q-values, and we show this recurrence to be correct and convergent to the value iteration algorithm in the fixed weight vector case. Many standard RL algorithms in the literature can be seen as limiting cases of our more general algorithm. While the worst-case complexity of our general algorithm is exponentially higher than that of fixed-$\overrightarrow{w}$ cases, it not only solves all the fixed-$\overrightarrow{w}$ cases but also determines which cases are worth solving. We also give some constraints and techniques that can help reduce the complexity.

3. Convex Hull Value Iteration

In this section, we introduce the problem definition in the context of a traditional MDP setting and our approach and algorithm.

3.1. Preliminaries and Notation

Our approach is based on an MDP which is a tuple $(S, A, T, \gamma, \overrightarrow{r})$, where S is a finite set of N states, $A = \{a_1, \ldots, a_k\}$ is a set of k actions, $T = \{\mathrm{P}_{sa}(s')\}$ is the set of state transition probabilities ($\mathrm{P}_{sa}(s')$ is the transition probability of going to state $s' \in S$ by taking action $a \in A$ from state $s \in S$), $\gamma \in [0, 1)$ is the discount factor, and $\overrightarrow{r} : S \times A \mapsto \mathbb{R}^d$ is the reward function giving d-component reward vector $\overrightarrow{r}(s, a)$. This differs from the standard formulation only in that reward now comes as a vector.

A policy, π, is the map $S \mapsto A$, and the value function for any policy π, evaluated at some state s_i is the vector

$$\overrightarrow{V}^{\pi}(s_i) = \mathbb{E}[\overrightarrow{r}(s_i, a_i) + \gamma \overrightarrow{r}(s_{i+1}, a_{i+1}) + \ldots |\pi] \quad (1)$$

where the expectation is over the distribution of the state and reward sequence $(s_i, \overrightarrow{r}_i, s_{i+1}, \overrightarrow{r}_{i+1}, \ldots)$, that is obtained on executing the policy π starting from the state s_i. The Q-function is the vector

$$\overrightarrow{Q}^{\pi}(s, a) = \mathbb{E}_{\overrightarrow{r}(s,a), s' \sim P_{sa}} \left[\overrightarrow{r}(s, a) + \gamma \overrightarrow{V}^{\pi}(s') \right] \quad (2)$$

where $\overrightarrow{r}(s, a), s' \sim P_{sa}$ means that the expectation with respect to s' and $\overrightarrow{r}(s, a)$ distributed according to P_{sa}. The optimal Q function for a weight $\overrightarrow{w}$ is $Q^*_{\overrightarrow{w}}(s, a) = sup_{\pi} \overrightarrow{w} \cdot \overrightarrow{Q}^{\pi}(s, a)$.

3.2. Approach: Convex Hulls

Given some $\overrightarrow{w}$, the resulting reward for taking an action is $r(s, a) = \overrightarrow{w} \cdot \overrightarrow{r}(s, a)$. This gives us the following recurrence for optimal Q-values, which is exactly equivalent to the equation for a single reward component:

$$Q_{\overrightarrow{w}}(s, a) = \mathbb{E} \left[\overrightarrow{w} \cdot \overrightarrow{r}(s, a) + \gamma \max_{a'} Q_{\overrightarrow{w}}(s', a') | s, a \right]$$

We can solve this recurrence directly, or we can use it to get converging approximations to the optimal value function—this gives rise to the value iteration method, Q-learning, and so on.

An alternative view is that each possible policy gives a different expected reward $\overrightarrow{Q}(s, a)$, and we simply want to select a policy by maximizing the dot product of this with $\overrightarrow{w}$. For a fixed $\overrightarrow{w}$, only one such $\overrightarrow{Q}(s, a)$ can be optimal, but in general we might care about any $\overrightarrow{Q}$s that are maximal for some $\overrightarrow{w}$. But this set of Q-values that are extrema is exactly the convex hull of the Q-values! This allows us to use standard convex hull operations to pare down the set of points we consider and gives rise to the following proposition.

Proposition 1. *The convex hull over Q-values contains the optimal policy over the average expected reward $r(s, a) = \overrightarrow{w} \cdot \overrightarrow{r}(s, a)$ for any $\overrightarrow{w}$.*

To make this operational and derive an algorithm that maintains all optimal policies for any weight vector $\overrightarrow{w}$, we need a few definitions for relevant operations on the convex hull.

We write $\overset{\circ}{Q}(s, a)$ to represent the vertices of the convex hull of possible Q-value vectors for taking

action a at state s. We then define the following operations on convex hulls which will be used to construct our learning algorithm.

Definition 1. Translation and scaling operations

$$\overrightarrow{u} + b\overset{\circ}{Q} \equiv \{\overrightarrow{u} + b\overrightarrow{q} : \overrightarrow{q} \in \overset{\circ}{Q}\} \tag{3}$$

Definition 2. Summing two convex hulls

$$\overset{\circ}{Q} + \overset{\circ}{U} \equiv \text{hull}\{\overrightarrow{q} + \overrightarrow{u} : \overrightarrow{q} \in \overset{\circ}{Q}, \overrightarrow{u} \in \overset{\circ}{U}\} \tag{4}$$

Definition 3. Extracting the Q-value *To extract the best Q-value for a given $\overrightarrow{w}$, we perform a simple maximum:*

$$Q_{\overrightarrow{w}}(s,a) \equiv \max_{\overrightarrow{q} \in \overset{\circ}{Q}(s,a)} \overrightarrow{w} \cdot \overrightarrow{q} \tag{5}$$

Given these definitions, we are now ready to illustrate the basic algorithm.

3.3. Convex Hull Value Iteration Algorithm

Our algorithm extends the single-$\overrightarrow{w}$ case (which is the standard expected discounted reward framework (Bellman, 1957)) into the following recurrence:

$$\overset{\circ}{Q}(s,a) = \mathbb{E}\left[\overrightarrow{r}(s,a) + \gamma\, \text{hull} \bigcup_{a'} \overset{\circ}{Q}(s',a') | s,a\right] \tag{6}$$

That is, instead of repeatedly backing up maximal expected rewards, we back up the set of expected rewards that are maximal for some $\overrightarrow{w}$. While the expectation over hulls looks awkward, it is the natural equivalent of an expectation of maxima, and it arises for the same reason. We must take an expectation over s', but once in s', we can choose the best action, no matter what our $\overrightarrow{w}$. The expectation's computation can be broken down, in the usual way, into the scalings and sums we have already defined.

This leads us to define Algorithm 1, which extends the value iteration algorithm (Bellman, 1957) to learn optimal Q-values for all possible $\overrightarrow{w}$. A proof of its correctness is given in Section 8.

Algorithm 1 Value iteration algorithm modified from that of Bellman (1957)

Initialize $\overset{\circ}{Q}(s,a)$ arbitrarily $\forall s, a$
while not converged **do**
  for all $s \in S$, $a \in A$ **do**
    $\overset{\circ}{Q}(s,a) \leftarrow \mathbb{E}[\overrightarrow{r}(s,a)$
      $+\gamma\, \text{hull} \bigcup_{a'} \overset{\circ}{Q}(s',a') | s,a]$
  end for
end while

return $\overset{\circ}{Q}$

4. Related Work

There is now a body of work addressing multi-reward reinforcement learning. There have been algorithms that assume a fixed ordering between different rewards, such as staying alive and not losing food (Gabor et al., 1998), techniques based on formulating the multiple reward problem as optimizing a weighted sum of the discounted total rewards for multiple reward types (Feinberg & Schwartz, 1995), and techniques that decompose the reward function into multiple components which are learned independently (with a single policy) (Russell & Zimdars, 2003). In all these cases, the preference over rewards is assumed to be fixed and time-invariant. In a slightly more flexible formulation, Mannor and Shimkin (2004) take multiple reward components and perform learning that results in expected rewards lying in a particular region of reward space.

More recently, (Natarajan & Tadepalli, 2005) formulate the multiple reward RL problem as we do, using a weighted expected discounted reward framework, and they store both the currently best policy and its Q-values as vectors. When priorities change dynamically (as reflected in changes in the weight vector), the agent can calculate new reward scalars from the vectors and thus start from the Q-values of the best policy learned so far rather than resetting entirely. As far as we are aware, none of the techniques proposed tackle the general case of learning optimal policies for all linear preference assignments over the multiple reward components.

4.1. Relation to POMDPs

Our problem, and hence its solution, is closely related to the standard partially observable Markov

decision process (POMDP) formulation. In a POMDP, we have a model of both observed and unobserved variables and use Bayesian reasoning to infer a joint distribution over the hidden variables. Then, we must choose an optimal action based on both the observed state and the continuous beliefs. (Kaelbling et al., 1998)

Consider the POMDP shown in Figure 3; here, the reward depends on an unobserved multinomial random variable, so $\mathbb{E}[r] = \sum_i \mathrm{P}(w = i) r_i$. If we define $\mathrm{P}(w_t | w_{t-1})$ to be the identity, the distribution of w will not change with t. Then, the expected reward depends linearly on our prior distribution over w, and the dual of the usual POMDP maximum-hyperplane algorithm corresponds to a convex hull operation over reward components. It is thus possible to write our multiple-reward problem as a POMDP problem. This suggests a natural route to extend our algorithm to operate on POMDPs. It remains future work, however, to see if the approximation algorithms used for solving POMDPs can yield useful results in our domain.

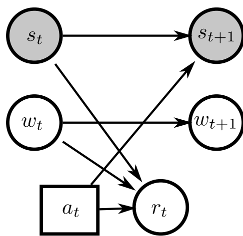

Figure 3. A POMDP formulation of multiple reward components

5. Complexity

This algorithm relies on four convex hull operations, whose complexity we will analyze in terms of the number of points on the hull, n; in the limit, this number converges to the number of optimal policies in the environment. We must both scale (by probabilities and discounts) and translate (by rewards) our convex hulls; these operations only require touching every point once, resulting in a complexity of $O(n)$. We must also merge two or more convex hulls. This takes time at most $O((kn)^{\lfloor d/2 \rfloor})$ if $d > 3$, where k is the number of hulls involved, n is the number of points in each hull, and d is the dimension (number of reward components) (Clarkson & Shor, 1989). Finally, we must add two convex hulls. If done naively by adding all pairs of points and taking a hull, this takes time at most $O(n^{2\lfloor d/2 \rfloor})$. All these operations must be performed whenever we back up Q-values, so we multiply the complexity of ordinary reinforcement learning by $O(n^{2\lfloor d/2 \rfloor})$. (However, in the $d = 2$ and $d = 3$ cases, there are efficient ways to perform these operations.)

In the long run, the number of points on each convex hull, n, must converge to a limit as the Q-values converge to their optimal values. Eventually, there will be exactly one point on each convex hull for each optimal policy. However, in the short term, the number of short-range policies we might have to track might be much lower or even higher. Also, the number of optimal policies n depends on the environment in a complicated way, with the worst case being that all policies ($|A|^{|S|}$ of them) may be optimal for some weight vector.

5.1. Reducing the Complexity

The complexity result of our algorithmic modifications is an exponential blowup with the number of reward components. There are a few main ways of tackling this. The first is to simply restrict the number of reward components; with only, say, 5 or fewer, this additional computation is likely not to be an undue burden. In practice, there are currently very few problems studied with more reward components than this.

When we must handle a high-dimensional problem, we can reduce the complexity by applying constraints on the weight vectors that we might optimize for. Given the geometric nature of our approach, if we have knowledge about the directions of allowable vectors, such as $\overrightarrow{a} \cdot \overrightarrow{w} > 0$, then we can simply take a partial convex hull. This will, on average, reduce the complexity of the convex hull computation by half. So, if we know that all d elements of $\overrightarrow{w}$ must be positive, then we can write that as d such constraints to divide the convex hull complexity by 2^d.

In addition, the convergence of Q-values means that we are essentially performing the same convex hull operations again and again; this means that

we might be able to reuse the information from the last iteration. The idea is to annotate each point with a "witness", or proof of its status: if a point is not on the convex hull, then we note down a set of faces that enclose it, and if it is on the hull, we note down a direction in which it is the extremum. Then, on the next iteration, when these points have moved slightly and we must compute a convex hull again, we can simply check these proofs (in at most $O(n^2)$ time). If all the proofs are correct, then our convex hull remains correct and the locations of the points have moved only slightly. On the other hand, if any proof is violated, we can simply rebuild the convex hull in the ordinary, expensive way. In the limit as the Q-values and policy converge, the policy must stop changing, so this trick may greatly reduce the complexity of refining Q-values.

6. Example Application: Resource Gathering

In order to demonstrate the application of this method, we have tested it on a resource-collecting problem similar to that of many strategy games. We model this as a resource-collecting agent moving (in the 4 cardinal directions) around in a grid environment shown in Figure 4, starting from the home base, labelled H. Its goal is to gather resources and take them back to the home base. If it reaches location R1, it then picks up resource 1, and at R2 it gets resource 2; it can carry both at the same time. When the agent returns to H, it receives a reward for each resource it brings back. Also, if it steps on one of the two enemy spaces, labeled E1 and E2, with a 10% probability it will be attacked, receiving a penalty and resetting to the home space, losing all it carries. Its reward space is then [enemy, resource1, resource2], so it can get a penalty of [-1,0,0] for being attacked, or a reward of [0,1,0], [0,0,1], or [0,1,1] for bringing back one or both resources. We use a discounting rate of $\gamma = 0.9$.

Depending on the relative values of the resources and attack, the agent may find different policies to be valuable. The convex hull of values starting at H, $\overset{\circ}{V}(H)$, is shown in Figure 5. The points on the hull correspond to optimal policies, described in Table 1; each policy is valid for some range of preferences $\overrightarrow{w}$, which are shown in Figure 6.[1]

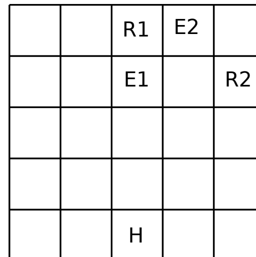

Figure 4. A resource-collection domain

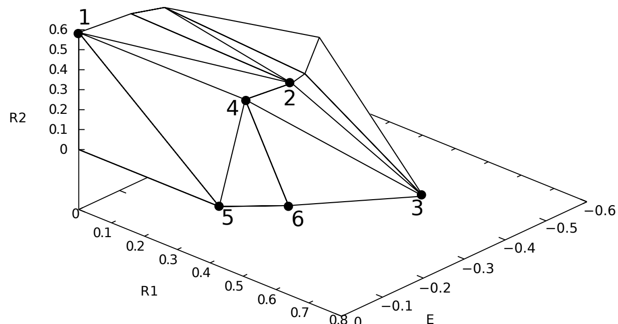

Figure 5. Optimal rewards in the resource-collection domain

#	**policy**
1	Go directly to R2, dodging Es
2	Go to both Rs, through both Es
3	Go to R1, through E1 both ways
4	Go to both Rs, dodging E1 but through E2
5	Go to R1, dodging all Es
6	Go to R1, going through E1 only once

Table 1. The optimal policies for the example domain

7. Extensions and Current Work

This same convex-hull technique can be used with other RL algorithms, such as the temporal difference learning algorithm. The critical thing to recall is that because we are learning more than one

[1]We do not show the ranges of policies optimal where the values of the rewards are less than 0 ($w_i < 0$); these policies, while sometimes interesting, are not valuable for the task.

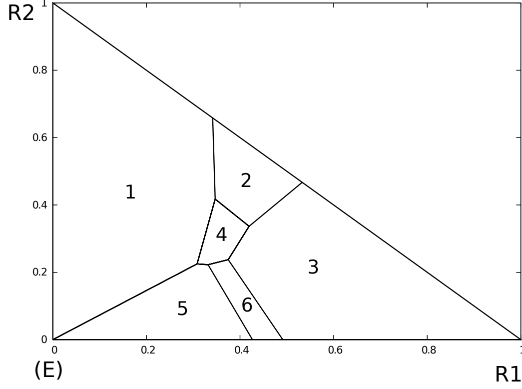

Figure 6. Regions of preference space in which policies are optimal. Axes are reward components $R1$ and $R2$; the enemy weight is $E = 1 - R1 - R2$.

policy at once, we can use only off-policy learning algorithms.

Our solution can also be used for inferring the preference function from observation data. This is closely related to the inverse reinforcement learning problem (Ng & Russell, 2000; Abeel & Ng, 2004). The basic idea behind inverse reinforcement learning is to use observed behavior to infer weights from a user that can then be used to find optimal policies. In our case, the method for learning all policies at once can also be used in reverse to learn the range of reward weights that an agent must have. If we assume that an agent we observe is rational and uses a policy that is optimal for its reward weights, then we can use our observations of the agent to infer its reward weights. We simply repeatedly observe its choice of action a and use our knowledge of $\mathring{Q}(s,a)$ to identify which values of $\overrightarrow{w}$ are consistent with that action. Then, we take the intersection of the constraints.

The multi-criterion RL approach also allows us to examine reward at different time scales. Instead of having a single discounting factor γ, we could have a discounting factor γ_i for each component. This allows us to use a sum of exponentials with different time constants to approximate non-exponential discounting rates, which are helpful in explaining the preferences of humans (Ainslie, 2001). With our convex hull method, we can find what policies are optimal for a whole range of discounting rates.

8. Appendix: Proof of Correctness

We prove that $\forall \overrightarrow{w}$ Algorithm 1 gives the optimal policy by reducing the recurrence to the standard value iteration recurrence for any $\overrightarrow{w}$. First, recall the basic recurrence of our algorithm, Equation 6.

$$\mathring{Q}(s,a) \leftarrow \mathbb{E}\left[\overrightarrow{r}(s,a) + \gamma \operatorname{hull} \bigcup_{a'} \mathring{Q}(s',a') | s,a\right]$$

Now apply Equation 5 to the both sides (to extract the optimal value for $\overrightarrow{w}$):

$$\begin{aligned} Q_{\overrightarrow{w}}(s,a) \quad \leftarrow \quad & \max\{\overrightarrow{w} \cdot \overrightarrow{q} : \overrightarrow{q} \in \mathbb{E}\Big[\overrightarrow{r}(s,a) \\ & +\gamma \operatorname{hull} \bigcup_{a'} \mathring{Q}(s',a') | s,a\Big]\}. \end{aligned}$$

Next, apply the definition of an expectation

$$\begin{aligned} \leftarrow \max \quad & \{\overrightarrow{w} \cdot \overrightarrow{q} : \overrightarrow{q} \in \textstyle\sum_{s',\overrightarrow{r}(s,a)} \mathrm{P}(s', \overrightarrow{r}(s,a)|s,a) \\ & \cdot \left(\overrightarrow{r}(s,a) + \gamma \operatorname{hull} \textstyle\bigcup_{a'} \mathring{Q}(s',a')\right)\}, \end{aligned}$$

then use Equations 3 and 4 and rewrite

$$\begin{aligned} \leftarrow \quad & \max\{\overrightarrow{w} \cdot \overrightarrow{q} : \overrightarrow{q} \in \operatorname{hull}\Big\{ \\ & \sum_{i,\overrightarrow{r}(s,a)} \mathrm{P}(s'_i, \overrightarrow{r}(s,a)|s,a) \left(\overrightarrow{r}(s,a) + \gamma \overrightarrow{q}'_{s'_i}\right) \\ & : \overrightarrow{q}'_{s'_1} \in \operatorname{hull} \bigcup_{a'} \mathring{Q}(s'_1, a'), \ldots \Big\}\}. \end{aligned}$$

$$\begin{aligned} \leftarrow \quad & \max\Big\{\overrightarrow{w} \cdot \sum_{i,\overrightarrow{r}(s,a)} \mathrm{P}(s'_i, \overrightarrow{r}(s,a)|s,a) \\ & \cdot \left(\overrightarrow{r}(s,a) + \gamma \overrightarrow{q}'_{s'_i}\right) : \\ & \overrightarrow{q}'_{s'_1} \in \operatorname{hull} \bigcup_{a'} \mathring{Q}(s'_1, a'), \ldots \Big\} \end{aligned}$$

$$\begin{aligned} \leftarrow \quad & \max\Big\{\mathbb{E}\Big[\overrightarrow{w} \cdot \overrightarrow{r}(s,a) \Big| s,a\Big] \\ & +\gamma \sum_i \mathrm{P}(s'_i|s,a) \overrightarrow{w} \cdot \overrightarrow{q}'_{s'_i} : \\ & \overrightarrow{q}'_{s'_1} \in \operatorname{hull} \bigcup_{a'} \mathring{Q}(s'_1, a'), \ldots \Big\}. \end{aligned}$$

Pull $\overrightarrow{r}(s,a)$ (added independently to the entire set) and γ (non-negative) out of the maximum.

$$\begin{aligned} \leftarrow \quad & \mathbb{E}[\overrightarrow{w} \cdot \overrightarrow{r}(s,a)|s,a] \\ & +\gamma \max\Big\{ \sum_i \mathrm{P}(s'_i|s,a)\overrightarrow{w} \cdot \overrightarrow{q}'_{s'_i} : \\ & \overrightarrow{q}'_{s'_1} \in \mathrm{hull} \bigcup_{a'} \mathring{Q}(s'_1,a'), \ldots \Big\} \end{aligned}$$

But the max of a sum over different sets is the sum of the sets' maxima, which we simplify.

$$\begin{aligned} \leftarrow \quad & \mathbb{E}[\overrightarrow{w} \cdot \overrightarrow{r}(s,a)|s,a] + \\ & \gamma \sum_i \mathrm{P}(s'_i|s,a) \max\Big\{ \overrightarrow{w} \cdot \overrightarrow{q}'_{s'_i} : \\ & \overrightarrow{q}'_{s'_i} \in \mathrm{hull} \bigcup_{a'} \mathring{Q}(s'_i,a') \Big\} \end{aligned}$$

$$\begin{aligned} \leftarrow \quad & \mathbb{E}[\overrightarrow{w} \cdot \overrightarrow{r}(s,a)|s,a] \\ & +\gamma \sum_i \mathrm{P}(s'_i|s,a) \max\Big\{ \overrightarrow{w} \cdot \overrightarrow{q}'_{s'_i} : \\ & \overrightarrow{q}'_{s'_i} \in \mathring{Q}(s'_i,a'), a' \in A(s'_i) \Big\} \end{aligned}$$

$$\begin{aligned} \leftarrow \quad & \mathbb{E}[\overrightarrow{w} \cdot \overrightarrow{r}(s,a)|s,a] \\ & +\gamma \sum_i \mathrm{P}(s'_i|s,a) \max_{a'} \max_{q'_{s'_i} \in \mathring{Q}(s'_i,a')} \overrightarrow{w} \cdot \overrightarrow{q}'_{s'_i} \end{aligned}$$

But we re-order the maxima and rewrite an expectation, and so we recover our recurrence for a single $\overrightarrow{w}$.

$$Q_{\overrightarrow{w}}(s,a) \leftarrow \mathbb{E}\left[\overrightarrow{w} \cdot \overrightarrow{r}(s,a) + \gamma \max_{a'} \mathring{Q}_{\overrightarrow{w}}(s',a') \middle| s,a \right].$$

Given a $\overrightarrow{w}$, at any point in the algorithm, this gives the same Q-value as ordinary value iteration. Therefore, the proof of convergence for the value iteration algorithm applies to our method, and our method converges exactly as quickly as ordinary value iteration (for every $\overrightarrow{w}$).

References

Abeel, P., & Ng, A. (2004). Apprentice learning via inverse reinforcement learning. *Proc. ICML-04*.

Ainslie, G. (2001). *Breakdown of will*. Cambridge, Massachusetts: Cambridge University Press.

Bellman, R. E. (1957). *Dynamic programming*. Princeton: Princeton University Press.

Clarkson, K. L., & Shor, P. W. (1989). Applications of random sampling in computational geometry, II. *Discrete and Computational Geometry*, *4*, 387–421.

Feinberg, E., & Schwartz, A. (1995). Constrained markov decision models with weighted discounted rewards. *Mathematics of Operations Research*, *20*, 302–320.

Gabor, Z., Kalmar, Z., & Szepesvari, C. (1998). Multi-criteria reinforcement learning. *Proc. ICML-98*.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*.

Mannor, S., & Shimkin, N. (2004). A geometric approach to multi-criterion reinforcement learning. *Journal of Machine Learning Research*, 325–360.

Natarajan, S., & Tadepalli, P. (2005). Dynamic preferences in mult-criteria reinforcement learning. *Proc. ICML-05*. Bonn, Germany.

Ng, A., & Russell, S. (2000). Algorithms for inverse reinforcement learning. *Proc. ICML-00*.

Russell, S., & Zimdars, A. (2003). Q-decomposition for reinforcement learning agents. *Proc. ICML-03*. Washington, DC.

Sutton, R., & Barto, A. (1998). *Reinforcement learning: An introduction*. Cambridge, Massachusetts: The MIT Press.